NAME

aps − APS driver script

SYNOPSIS

aps [*init* | *start* | *stop* | *kill* | *term*]

aps *repro* [*list* | *dir*] [area1 area2 ...]

DESCRIPTION

The `aps` script is the driver for the Automated Processing System. It is also the user's administrative interface to the Automated Processing System. From this script the user controls the system. Additionally, this script is used for reprocessing data.

init

The *init* command is used to start up the APS driver. The prompt will return immediately, but the script has restarted itself in the background and will enter an infinite loop.

When initialized the APS driver will create three files: `.aps.pid` , `.aps.process`, and `.aps.log`. The first file contains the processing ID of the `aps` script. It is used to terminate and/or kill a running APS driver. The second file is used to tell the APS driver whether to process any data. It is checked for existance once a second (by default). If non-existant, no data will be processed. It provides the user a method to temporarily halt data processing without having to terminate the APS driver (see the *start* and *stop* commands). The last file is a logging file. The APS driver will write messages about the files it has processed to this file.

term

The *term* command is used to terminate the APS driver whose PID is in the file `.aps.pid`. This may take a few moments or up to a few hours to complete as the APS driver will wait for any children processes to complete first.

kill

The *kill* command is used to immediately kill the APS driver whose PID is in the file `.aps.pid`. Try *term* first and if it does terminate fast enough for you, then use this. The children of the APS driver *may* still be running, however. This is usually called by the `rc.aps` script, prior to a system shutdown.

start

The *start* command creates the `.aps.process` file which is used to signal the APS driver to proceed with data processing. Usually this is only called if `stop` had been previously run.

stop

The *stop* command deletes the `.aps.process` file which is used to signal the APS driver to stop processing data. When the file is non-existant the APS driver will continue to poll the `in` directory, but will never process the data.

One use of the two commands *start* and *stop* is within `cron` to prevent the APS from running (and taking up computer resources) during the work day. This was its original intent, though the APS is now usually run on a dedicated system.

repro

The *repro* option is the preferred way to reprocess data. The user must either provide the program with a list of data files to reprocess stored in a UNIX text file or (2) provide a directory path for which the files are contained.

If option a list file is used, it is best to use full pathnames in it. Each file to be reprocessed must reside on a single line in the text file. Given the “list” file, the APS driver will execute each script in the `areas` directory sequentially on each file. The files listed must include the full pathname. They may be compressed using `gzip` or `compress`. If any files are compressed, then the path to the local copy of the gzip program must be defined by the keyword CmpGzip found in the `aps.conf` file. (Normally, it is defined during the installation of the APS software.) The *repro* option will automatically place an (uncompressed) copy of each input file in the `$AUTO_PROC` directory before processing. The copy is removed automatically. *The original input files are not touched.*

The user can also specify which of the `areas` scripts are used in the reprocessing. For those area names listed on the command line that have relative paths, the `$AREAS_PROC` directory will automatically be searched for the given area script. If the user does not specify which areas to process, then *all* areas in the `$AREAS_PROC` directory are used.

If the environmental variable `$REPROC_MAIL` is set, then an e-mail message will be issued to the user defined in `$REPROC_MAIL` for each file when it is completed and a final message when all files have been reprocessed.

For example, suppose we wish to reprocess the entire 1999 year of SeaWiFS data for the MissBight and GulfOfMexico regions. Then we might do the following:

```
$ dir=/rs/lvl1/seawifs/hrpt/HNAV/1999
$ find $dir -name "S*gz" -print | sort > ˜/repro.list
$ export REPROC_MAIL=username@local.domain
$ ˜/aps_v2.6/bin/aps repro ˜/repro.list SwfMissBight SwfGulfOfMexico
Processing these regions:
    /people/apsdev/aps_v2.6/areas/SwfMissBight
    /people/apsdev/aps_v2.6/areas/SwfGulfOfMexico
Working in S1999091174814.L1A_HNAV ...
SwfMissBight : checking if file covers MissBight ... yes.
SwfMissBight : converting L1 to L2 ... done.
SwfMissBight : converting L2 to L3 ... done.
```

output continues ...

ENVIRONMENTAL VARIABLES

The environmental variables in this section are required to define the directory structure used by the APS system. Many of the environmental variables have defaults if the normal directory structure is followed. They are defined in the configuration file `aps.conf` located in the `bin` directory. Modify the configuration file if you need to set these to some other values.

AUTO_DIR
: The top level directory for the APS system.

AUTO_BIN
: The location of executables for the APS system. Defaults to `$AUTO_DIR/bin`.

AREAS_PROC
: The directory containing executable scripts for processing data received by the APS system. These scripts will be passed a single argument being the name of the input file with no pathname. Defaults to `$AUTO_DIR/in`.

AUTO_RAW
: The directory which the aps script will continuously poll for new data to process. This directory will probably have write permissions for to allow other users to write data to this directory. Defaults to `$AUTO_DIR/in`.

AUTO_PROC
: The directory where all the data processing will be performed. Defaults to `$AUTO_DIR/work`.

AUTO_OUT
: The directory where all the final products will be moved. Defaults to `$AUTO_DIR/out`.

AUTO_ERROR
: The directory where all the files will be moved, when there is an error during the processing. Defaults to `$AUTO_DIR/err`.

AUTO_DATA
: The directory where all the data files exist. Defaults to `$AUTO_DIR/data`.

CONFIGURATION VARIABLES

The configuration variables are sourced from the APS configuration file. This file resides in the `$AUTO_BIN` directory.

apsPollingTime
: This variable controls the number of seconds the APS driver will sleep before examining the `$AUTO_RAW` directory. The default is sixty seconds.

apsLogFile
: Name of the file to receiving all logging information. The default is `$AUTO_BIN/.aps.log`.

apsPIDFile
: Name of file to receive the PID of the running APS. This file is used to terminate and/or kill the APS system. It is also used as a "lock" file. If the user attempts to initialize the APS, the attempt will fail if this file exists. This variable defaults to `$AUTO_BIN/.aps.pid`.

apsProcessFile
: Name of the file used as an "on/off" switch. This file allows the user to temporarily stop/start the APS processing. It will only effect the `next` file. That is, if the APS is currently processing a file, it will continue to do so. However, any remaining files will not be processed. The existance of this file will turn the APS "on" while its non-existance will turn the APS "off" This file defaults to `$AUTO_BIN/.aps.process`.

apsPreProcess
: Name of a script to run prior to running the areas scripts for each file found in the `in` directory. This script will only be executed when a file is found and the script has execution permissions. The version included with APS is used to convert a SeaWiFS Level−0 file to Level−1A. To stop apsPreProcess from running, remove its execution permissions.

apsPostProcess

Name of a script to be run after the areas scripts have been run for a given file. This script will be run giving it the name of the file. It will only run when a file is to be processed and the script has execution permissions. At NRL the script is used to move the input file to the archive system. To stop apsPostProcess from running, remove its execution permissions.

apsCronProcess

Name of an executable script to be run for each polling cycle. This can be used whenever some processes need to be on a regular basis. The apsCronProcess included with APS is used to check for incoming data from three regional centers.

FILES

```
$AUTO_BIN/aps.conf
$AUTO_BIN/.aps.pid
$AUTO_BIN/.aps.process
$AUTO_BIN/.aps.log
```

SEE ALSO

aps_intro(1), **aps.conf**(5)

NAME

apsScripts – Bourne shell scripting functions.

DECRIPTION

This file contains a series of Bourne shell script functions which are useful for the writing scripts needed to process satellite data.

Note: All shell variables are "global", so the `apsScripts` functions uses the convention that all variables that are intended for use *outside* the script function start with a capital letter. All lowercase variables are "local". Unfortunately, this does not prevent them from being overwritten by other script functions called within a particular script function.

`apsAddToDatabase file directory`

The script function `apsAddToDatabase` is intended to be used by shell scripts to move `file` to a simple directory–type database.

The function will exit if `file` is not a regular file, or if path given as directory already exists but is not actually a directory. If the variable `DataPerms` is defined, then `chmod` is run to change the its permissions.i Next, the file is copy to the `$SIPRNETOutDir` if defined. The directory argument is now checked. If the given directory path does not exist, it is created. Next, the file is compressed using the command as defined by the `$CmpOpt` variable. If the variable does not exist, it defaults to the UNIX `touch` command (that is, no compression is performed.) Then `directory` is checked for existence and is created (with any missing limbs) if it does not. Finally, the (compressed) file is moved to the directory.

`apsAddToWWW file www_dir`

The script function `apsAddToWWW` is intended to be used by shell scripts to move the file to a directory controlled by a Web server. If the `www_dir` is of the form `hostname:directory`, i.e., it contains a colon “:”, then `file` is remotely copied (using `rcp`) to that directory. Otherwise it is simply moved to that directory using `mv`. If `www_dir` does not exist, the `apsMakeDir` script function is called to create it.

If `file` is copied to a remote machine, it’s permissions are changed to those indicated by the variable `WWWPerms` (if defined) before being copied. Otherwise, the `DataPerms` is used (again only if it is defined).

`apsAppend var str`

Appends to the string `str` to the variable `var`.

`apsCheckDir file`

The script function `apsCheckDir` is intended to be used by shell scripts to determine if the argument represents an directory or not. This script calls `apsFatalErr` (which terminates the script) if the directory does not exist or is unreadable.

`apsCheckExec`

The script function `apsCheckExec` is intended to be used by shell scripts to determine if the argument represents an executable file or not. This script calls `apsFatalErr` (which terminates the script) if program is not a regular file or not executable.

`apsCheckFile file`

The script function `apsCheckFile` is intended to be used by shell scripts to determine if the argument represents an file or not. This script calls `apsFatalErr` (which terminates the script) if the directory does not exist or is unreadable.

`apsFindExec program`

This function searches through the colon-seperated directory list of the `PATH` environment variable to determine full pathname of the given program. If not found, `apsFatalErr` is called and the script is terminated.

`apsFatalErr arg1 arg2`

The script function `apsFatalErr` is intended to be used by shell scripts when a fatal error has occured. The arguments are passed to `apsLogMsg` and `exit 1` is called, thereby, terminating the script.

`apsInfo inFile zLat zLon`

The script function `apsInfo` is used to gain information about an input file. The satellite specific routine initialization (`avhInit`, `swfInit`, etc.) routine should be called prior to this function. These will define the satellite specific info program in the variable `$ApsInfo`.

If the variable `$DayLight` is not defined (set in `apsInit`), then the output variable `$Day` is set to 9 (unknown).

These variables are created by this function. These can (and are) used by other scripts to for such things as the default image data directory structure.

`Year`	Set to Year (YYYY) of data file
`DoY`	Set to Day of Year (DDD) of data file
`DoM`	Set to Day of Month (DD) of data file
`Month`	Set to Month (mmm) of data file
`Time`	Set to Time (hhmmss) of data file
`Hour`	Set to Hour (hh) of data file
`Min`	Set to Minute (mm) of data file
`Sat`	Set to Satellite ID (sss) of data file
`Day`	Set to 1 for Day, 2 for Night, or 3 for Day/Night or twilight

To compute `Day`, the variables `mapCLat`, `mapCLon`, `mapC1Lat`, `mapC1Lon`, `mapC2Lat`, `mapC2Lon`, `mapC3Lat`, `mapC3Lon`, `mapC4Lat`, and `mapC4Lon`, need to be defined. These are normally defined by `apsInit` by calling `apsMapInfo`.

`apsInit`

The script function `apsInit` is intended to be used by shell scripts to set up some shell variables used by other functions within the file apsScripts. These allow for system differences to be incapsulated in a single location. The use of variable names for executables also allows the user to replace an executable to a modified one for testing new algorithms with little change to the standard scripts.

These variables are set using the environmental variables that are usually defined in `aps.conf` file by a script executed as a child process of the APS driver process. In a user script, these may be overwritten before any call to `apsInit`.

`AutoBin`

Set to AUTO_BIN environment variable

`AutoData`

Set to AUTO_DATA environment variable

`DataBase`
: Set to DATA_BASE environment variable

`ImagBase`
: Set to IMAG_BASE environment variable

These variables are set by using the `apsFindExec` script function to search the `PATH` environmental variable.

`Remove`
: Set to fullpath of `rm` command with '`-f`' appended

`Move` Set to fullpath of `mv` command

`Copy` Set to fullpath of `cp` command

`RCopy` Set to fullpath of `rcp` command

`Chmod` Set to fullpath of `chmod` command

`Chgrp` Set to fullpath of `chgrp` command

`MkDir` Set to fullpath of `mkdir` command

`Touch` Set to fullpath of `touch` command

`Tar` Set to fullpath of `tar` command

`Cat` Set to fullpath of `cat` command

These variables define the various compression options known by the APS. Most are found by using the `apsFindExec` script function to search the `PATH` environmental variable.

`CmpOpt`
: Set to "none".

`CmpNone`
: Set to fullpath of `touch` command

`CmpNoneExt`
: Set to an empty space.

`CmpCompress`
: Set to fullpath of `compress` command with '`-f`' appended

`CmpUnCompress`
: Set to fullpath of `uncompress` command with '`-f`' appended

`CmpCompressExt`
: Set to `.Z`

`CmpGzip`
: Set to fullpath of `gzip` command with '`-f`' appended

`CmpGunzip`
: Set to fullpath of `gunzip` command with '`-f`' appended

`CmpGzipExt`
: Set to `.gz`

`LogFile`
: Set to `dev/null`

`DataPerms`
: Set to 644

`WWWPerms`
: Set to 644

`DayLight`
: Set to `AutoBin/daylight`

`FileFmt`
: Set to `AutoBin/filefmt`

`Gregor`
: Set to `AutoBin/gregor`

`Hdf` Set to `AutoBin/hdf`

`Maps` Set to `AutoBin/maps`

`ApsCatalog`
: Set to `AutoBin/apsCatalog`

`ApsSIPRNET`
: Set to `AutoBin/apsSIPRNET`

`MapFile`
: Set to `AutoData/maps.hdf`

`apsLock lockfile`

The script function `apsLock` is intended to be used by shell scripts to provide a locking mechanism when certain functions must only run atomically. Used in combination with the `apsUnlock` function.

`apsLogMsg arg1 arg2 ...`

The script function `apsLogMsg` is intended to be used by shell scripts to write time stamped messages to a log file. The arguments are passed to echo and redirected to the `LogFile` variable. If `LogFile` is not set, **apsLogMsg** as no effect.

`apsMakeDir directory`

The script function `apsMakeDir` is intended to be used by shell scripts to create the complete directory path, recusively. Each part of the directory is analyized for existence. Any part that does not exist is created.

If the directory path is of the form `hostname:directory` then the `Rsh` command will be used to attempt to create this directory on the remote system provided it has permissions.

Currently, this function cannot handle relative paths.

`apsMakeMine file`

The script function `apsMakeDir` is intended to be used by shell scripts to make the script user the owner of the file if not already owner. Additionally, it sets the permissions to 444 (read only).

`apsMapInfo`

This script function `apsMapInfo` makes successive calls to the `Maps` program to obtain infromation about the map. It creates the variables `mapSamples`, `mapLines`. Additionally, it extracts the lat/lon location of the four corner points and center to be used by apsInfo to determine if the file is day, night, or twilight.

`apsMkTemp variable file-prefix`

This script function is used to return a unique temporary filepath in variable

`apsSIPRNET file`

This script is used to copy the attributes from the file into another file with the same name but in the `$SIPRNET` directory for manual transfer by user.

`apsSetCompress name`

This script function will set the `opt` and `optExt` variables based on the input name. The input name can be one of ‘‘compress’’, ‘‘gzip’’, or ‘‘none’’. Otherwise, it defaults to `opt` to `touch` and `optExt` to a blank string.

`apsUnlock lockfile ...`

The script function `apsUnock` is intended to be used by shell scripts to provide a locking mechanism when certain functions must only run atomically. Used in combination with the `apsLock` function.

NAME

daylight – determine if sun is up for given time and place

SYNOPSIS

daylight year day hour min lat lon

DESCRIPTION

The program **daylight** is intended to be used by shell scripts when two courses of action must be made: one for night time and the other for daytime. The program will determine the elevation of the sun based on a location and time of day. If the elevation is greater than 20 degrees, the program exits with a status of 1 (day). Otherwise, the exit status is 0 (night).

The time must be given as a four digit year, day of the year, hour and minute of the day in UTC. The location is given by the latitude and longitude of the point of interest. These must given in decimal degrees ranging from (-90.0 to 90.0) and (-180.0 to 180.0) respectively.

OPTIONS

-e # Used to change the default value (20.0) for the elevation.

-v Use verbose mode.

--help Print out a small help page.

--version
Print out version of software and quit.

NAME

filefmt – determines format of file(s)

SYNOPSIS

filefmt file ...

DESCRIPTION

The program **filefmt** is intended to be used by shell scripts when it must determine the type of the file in question. The formats supported are most graphics file formats and several *remote sensing* file formats. The program reads information from the file to determine its type, and therefore, must have read permissions on the input file(s). If the format can be determined, the program prints to stdout: the filename, a tab, and the type as an ASCII string. See **FORMATS** below for a list of known types.

For the last (or only) file on the command line, a numeric code will also be set as the exit status of the program. This capability can be used by shell scripts to determine different courses of action depending on the file type.

OPTIONS

--help Print out a small help page.

--version
Print out version of software and quit.

FORMATS

The table below provides the numeric, ASCII string, and description of the formats known by this program.

Exit Code	*String*	*Description*
0	UNKNOWN	
1	PostScript	PostScript file.
2	GIF	GIF file.
3	JPEG	JPEG file.
4	SRF	Sun Raster file.
5	SGI	Silicon Graphic’s .rgb file
6	BMP	BMP file.
7	TIFF	Tagged Image File.
8	PBM	Portable BitMap File.
9	PGM	Portable Gray File.
10	PPM	Portable PixMap File.
11	PBM_RAW	Portable BitMap File (Raw).
12	PGM_RAW	Portable Gray File (Raw).
13	PPM_RAW	Portable PixMap File (Raw).
14	XBM	X BitMap File.
32	NASA/GSFC CZCS Level-1	CZCS CRTT Level-1 Format
33	NESDIS AVHRR 1b (LAC)	
34	NESDIS AVHRR 1b (GAC)	
35	Reserved	
36	Reserved	
37	Terascan HRPT Telemetry	Level-0 HRPT (AVHRR).

38	Terascan SWHRPT Telemetry	Level-0 swapped HRPT (SeaWiFS).
39	NASA SeaWiFS Level-0	NASA-defined Format.
40	SeaWiFS Level-1A Data	NASA-defined Format.
41	SeaWiFS Level-2 Data	NASA-defined Format.
42	SeaWiFS Level-2 Q/C Data	NASA-defined Format.
43	SeaWiFS Level-3 Binned Data	NASA-defined Format.
44	NESDIS AVHRR 1b (HRPT)	
46	SeaWiFS Level-1A Browse Data	NASA-defined Format.
47	SeaWiFS Level-2 Browse Data	NASA-defined Format.
48	SeaWiFS Level-3 Browse Data	NASA-defined Format.
49	SeaWiFS Level-3 Standard Mapped Image	NASA-defined Format.
50	Sensor Calibration Data	NASA-defined Format.
60	OCTS Level-1B Data	NASA-defined Format.
61	OCTS Level-1B Data	NADAQ-defined Format.
70	MOS Level-1B Data	NASA-defined Format.
75	POLDER Level-1B Data	NASA-defined Format.
100	MODIS Level-1B Data (1KM)	NASA-define Format.
101	MODIS Level-1B Data (500 meter)	NASA-defined Format.
102	MODIS Level-1B Data (250 meter)	NASA-defined Format.
103	MODIS Geolocation Data	NASA-defined Format.
104	MODIS Level-2 Cloud Mask Data	NASA-defined Format.
120	MODIS Level-1B Data (1KM)	Wisconsin-defined Format.

EXAMPLE

In this example, notice that the second command echo’s the return value of the file which is "40" because the file is a SeaWiFS Level-1A File.

```
$ filefmt S1999350182100.L1A_HNAV
S1999350182100.L1A_HNAV SeaWiFS Level-1A Data
$ echo $?
40
```

SEE ALSO

file(1)

NAME

gregor − converts from day-of-year to month/day and vice-versa.

SYNOPSIS

gregor year yday
−or−
gregor year month mday

DESCRIPTION

The program **gregor** will convert a day of the year into a month and day or vice versa. The year must be a four-digit year.

OPTIONS

-d n
Add *n* number of days to input date

-r
Compute the "Reynolds day" for given date. The "Reynolds day" is the Wednesday of each week and corresponds to file naming scheme used by Reynolds for his optimum interpolation sst values.

--help
Print out a small help page.

--version
Print out version of software and quit.

NAME

hdf − general manipulation functions on an HDF file.

SYNOPSIS

hdf file.hdf [*cat* | *copy* | *fattr* | *list*] [*other parameters*]

DESCRIPTION

The program **hdf** is used to manipulate HDF files (currently it only supports the SDS interface). With this program you can copy an SDS (with or without attributes) from one HDF file to another, dump an SDS to the screen, list all SDSs and file attributes in a format similar to NetCDF's Common Data Language (CDL).

The **file.hdf** parameter is the input file to perform the given action on. There will be a different series of parameters for each action taken. The actions that can be performed on an HDF file are: cat, list, copy.

ACTIONS

cat

The *cat* command is used to dump data from the given SDS to the screen. The data will be printed according to its type (char's as string, int's as integers, and float's as floating point numbers). There will be 16 columns across the page for 8− and 16−bit signed/unsigned integers; and 8 columns across the page for 32−bit signed/unsigned interger or 32− and 64−bit floating point numbers. Characters will be printed to 72 columns.

copy

The *copy* command is used to copy an SDS from one file to another.

To copy all file attributes from one HDF file to another file, use the comand:

hdf from.hdf copy -attr to.hdf

To copy all attributes from an SDS in one file to an SDS in another file, use the command:

hdf from.hdf copy -attr to.hdf from.sds to.sds

To copy only specified attributes from one SDS in one file to an SDS in another file, use the command:

hdf from.hdf copy -attr to.hdf from.sds to.sds attr1 attr2 ...

To copy the attributes from an SDS in one file to an SDS in another file renaming the attributes in the process, use the command:

hdf from.hdf copy -attr -rename to.hdf from.sds to.sds attr1 newattr1 attr2 newattr2 ...

To copy an SDS with its associated attributes to another file, use the command:

hdf from.hdf copy to.hdf from.sds to.sds sds1 sds2 ...

To copy the SDS without the associated attributes add the -noattrs option:

hdf from.hdf copy -noattrs to.hdf from.sds to.sds sds1 sds2 ...

To rename the SDS's during the copy:

hdf from.hdf copy [-noattrs] to.hdf from.sds to.sds sds1 new1 sds2 new2 ...

fattr

The *fattr* command is used to add or change a file attribute. The command has a single option −nt used to define the number type of the attribute. The default number type is "DFNT_CHAR8" normally used to append strings attributes. Currently, this option can not handle arrays.

To add an attribute called ’browseList’ with a value of ’chl_oc2,chl_oc4’, the user would run the command:

hdf in.hdf fattr browseList chl_oc2,chl_oc4

list

The *list* command is used to get an ASCII dump of the contents of the HDF file in a format similiar to the netCDF Common Data Language format. The user can also request the output to be in HTML format.

hdf from.hdf list [*type*]

The parameter *type* can be either ‘‘-ascii’’ or ‘‘-html’’. If type is not supplied or understood by **hdf**, then it defaults to ‘‘ascii’’.

sattr

The *sattr* command is used to add or change an SDS attribute. The command has a single option **−**nt used to define the number type of the attribute. The default number type is "DFNT_CHAR8" normally used to append strings attributes. Currently, this option can not handle arrays.

To add an attribute called ’comment’ with a value of ’very nice image’, the user would run the command:

hdf in.hdf sattr chl_oc4 comment very nice image

OPTIONS

--help Print out a small help page.

--version
Print out version of software and quit.

NAME

maps – interactive program to manipulate maps.

SYNOPSIS

maps – or **– maps [options] MapFile MapName**

DESCRIPTION

The program **maps** is an interactive program used to manipulate image maps. It is *not* intended to be very robust. It was originally designed to test the Aps_MapsXXX set of routines. But is also the only way to currently create and write an image map to a file for use in the areas scripts.

When the program is executed, the user is provided with a prompt to receive commands. The quick summary of the commands can be displayed using the ''help'' command. Below is a quick summary of the commands.

exit quit

Quits the program.

help

Prints a simple help menu. This list all the valid commands and one line on their purpose.

load

Adds any maps found in the user-specified file to the internal list.

save

Saves the internal list of maps to the user-specified file.

create

Creates a new map. This option will prompt the user for various parameters of the image map. Map projection 5 (Mercator) is the only one that has been extensively tested.

delete

Deletes the user-specified map from the internal list.

dist_xy

Compute the distance in meters between two points on a given map. The map projection must be set prior to using this option.

dump

Dump the current navigation structure. A debugging option.

list

Lists all the maps in the internal list.

current

Lists the currently set image map.

setmap
Sets the user selected map to the current image map.

show
Prints out the parameters for a user-specified image map.

toll x y
Used to convert the user specified image coordinates (x,y) to geographic coordinates (lon,lat).

toxy lon lat
Used to convert the user specified geographic coordinates (lon,lat) to image coordinates (x,y).

toij lon lat
Used to convert the user specified geographic coordinates (lon,lat) to map coordinates (i,j).

grid
Using the currently set image map, a world grid (lon,lat) is converted to image coordinates. The results are printed as a simple table.

dist_xy x1 y1 x2 y2
Computes the distance in meters between the two given image coordinates (x1,y1) and (x2,y2).

This program will be replaced by a more robust and GUI-oriented version some time in the future.

version
Prints out the version of this program.

OPTIONS

-c Print out the center longitude/latitude of a specified *image map*.

-C *n* Print out the cornet longitude/latitude of a specified *image map*. Where *n* is 0 for upper left corner, 1 for upper right corner, 2 for lower right corner, 3 for lower left corner.

-h Print out image height of specified *image map*.

-w Print out image width of specified *image map*.

--help Print out a small help page.

--version
Print out version of software and quit.

MAP PROJECTIONS

The following is a list of map projections supported by APS.

Number	Name
0	Geographic
1	Universal Transverse Mercator (UTM)

2	State Plane Coordinates
3	Albers Conical Equal Area
4	Lambert Conformal Conic
5	Mercator
6	Polar Stereographic
7	Polyconic
8	Equidistant Conic
9	Transverse Mercator
10	Stereographic
11	Lambert Azimuthal Equal Area
12	Azimuthal Equidistant
13	Gnomonic
14	Orthographic
15	General Vertical Near-Side Perspective
16	Sinusiodal
17	Equirectangular
18	Miller Cylindrical
19	Van der Grinten
20	(Hotine) Oblique Mercator
21	Robinson
22	Space Oblique Mercator (SOM)
23	Alaska Conformal
24	Interrupted Goode Homolosine
25	Mollweide
26	Interrupted Mollweide
27	Hammer
28	Wagner IV
29	Wagner VII
30	Oblated Equal Area
99	User defined

SPHEROIDS

The following is a list of spheriods supported by APS.

Number	Name
0	Clarke 1866
1	Clarke 1880
2	Bessel
3	International 1967
4	International 1909
5	WGS 72
6	Everest
7	WGS 66
8	GRS 1980
9	Airy
10	Modified Everest
11	Modified Airy
12	WGS 84
13	Southeast Asia
14	Australian National
15	Krassovsky
16	Hough
17	Mercury 1960

18	Modified Mercury 1968
19	Sphere of Radius 6370997 meters
20	Bessel 1841(Namibia)
21	Everest (Sabah & Sarawak)
22	Everest (India 1956)
23	Everest (Malaysia 1969)
24	Everest (Malay & Singapr 1948)
25	Everest (Pakistan)
26	Hayford
27	Helmert 1906
28	Indonesian 1974
29	South American 1969
30	WGS 60

ACKNOWLEDGEMENTS

The map projection software used by the APS came from the United States Geological Survey and is known as the General Cartographical Transformation Package. The APS uses version 2.0 of the C library.

EXAMPLES

For scripting purposes, the user might want to know the image dimensions of a an *image map*. To do that, we might use the following:

```
$ maps -w ˜/aps_v2.6/data/maps.hdf GulfOfMexico
2430
$ maps -h ˜/aps_v2.6/data/maps.hdf GulfOfMexico
1810
```

For the center and four corner points we have:

```
$ maps -c ˜/aps_v2.6/data/maps.hdf GulfOfMexico
-89.007407 25.061878
$ maps -C0 ˜/aps_v2.6/data/maps.hdf GulfOfMexico
-98.000000 31.000000
$ maps -C1 ˜/aps_v2.6/data/maps.hdf GulfOfMexico
-80.007407 31.000000
$ maps -C2 ˜/aps_v2.6/data/maps.hdf GulfOfMexico
-80.007407 18.811032
$ maps -C3 ˜/aps_v2.6/data/maps.hdf GulfOfMexico
-98.000000 18.811032
$ maps -C4 ˜/aps_v2.6/data/maps.hdf GulfOfMexico
invalid corner
```

See the Automated Processing Systems User’s Guide for an interactive example of this program’s usage.

SEE ALSO

Aps_MapsInit (3)